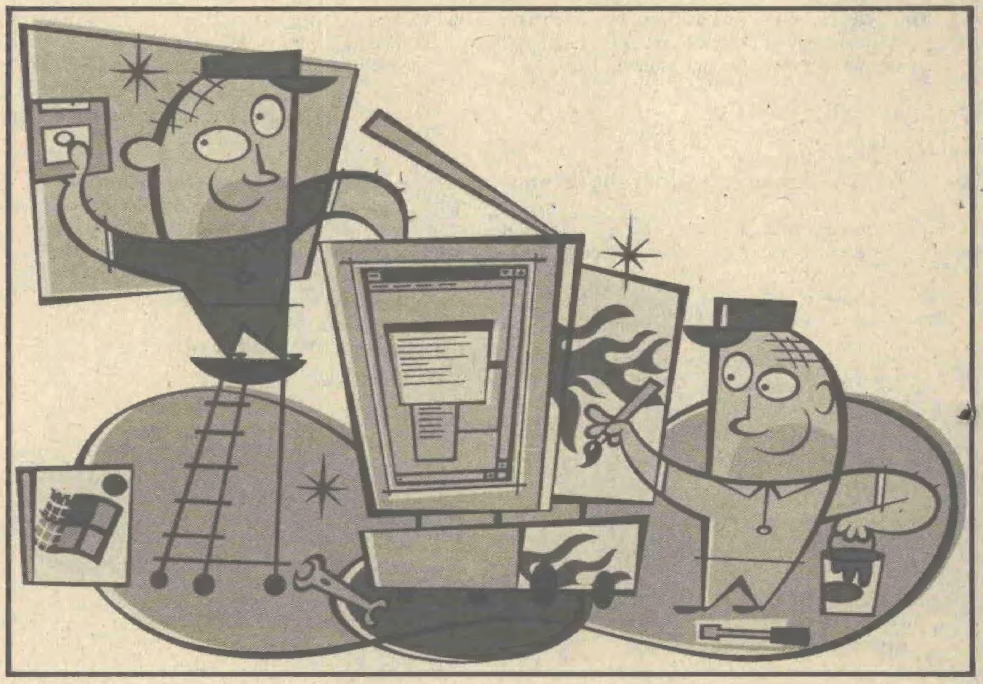
สรรค์ เสกขุนทด

เทคนิคการเขียนโปรแกรมบนวินโดวส์ ตอน เมสเสจ



เสนอเรื่องราวและเทคนิคเกี่ยวกับเมสเสจ ในการเขียนโปรแกรมบนวินโดวส์อย่างถูกต้องและมีประสิทธิภาพ

อาจจะเป็นไปได้ไหมว่าในอนาคตอันใกล้นี้ เรา (ประเทศไทย) จะต้องนำเข้าซอฟต์แวร์ต่างประเทศอีกเป็นจำนวนมาก จะมีหนทางใดบ้างที่จะช่วยลดการพึ่งพาต่างประเทศให้น้อยลง เพื่อว่าอุตสาหกรรมคอมพิวเตอร์ของไทยจะยืนหยัดสู้กับซอฟต์แวร์ต่างประเทศได้ และถ้าเป็นไปได้เราควรจะมีอุตสาหกรรมคอมพิวเตอร์เพื่อการส่งออก ไมโครซอฟต์วินโดว์และไมโครซอฟต์วินโดว์เอ็นทีกำลังเป็นมาตรฐานเข้าไปทุกขณะ ซอฟต์แวร์จำนวนมากกำลังถูกย้ายมายังวินโดว์ แล้วเราจะหยุดนิ่งอย่างนั้นหรือ?

การพัฒนาซอฟต์แวร์เป็นเรื่องที่ต้องอาศัยความละเอียดอ่อน และที่สำคัญข้อมูลจะต้องเพรียบพร้อม จึงจะได้ซอฟต์แวร์ที่มีคุณภาพ ดังนั้นคอลัมน์นี้จึงได้ก่อเกิดขึ้นมา โดยมีเนื้อหาจะมุ่งไปที่เทคนิคและความรู้เกี่ยวกับไมโครซอฟต์วินโดว์และไมโครซอฟต์วินโดว์เอ็นที แนวคิดและข้อมูลที่น่าสนใจจะได้ทยอยมาเสนอทีละหัวข้อ และในตอนแรกนี้เป็นเรื่องของ "เมสเสจ" ซึ่งถือเป็นเรื่องที่สำคัญมากในระบบวินโดว์

ว่าด้วยเรื่องเมสเสจ

ใครก็ตามที่เขียนโปรแกรมบนวินโดว์คงจะต้องคุ้นเคยกับคำว่า "เมสเสจ" หรือ "อีเวนต์" เป็นอย่างดี โดยเฉพาะช่วงที่พึ่งจะเริ่มต้นใหม่ กว่าจะปรับการเขียนโปรแกรมแบบควบคุมตั้งแต่ต้นจนจบมาเป็นการเขียนโปรแกรมแบบ "รอเมสเสจ" ได้ก็กินเวลานานทีเดียว

ดังนั้นในตอนแรกของคอลัมน์นี้จึงได้นำเสนอหัวเรื่องนี้ เพราะได้คำนึงแล้วว่าโครงสร้างการส่งเมสเสจเป็นหัวใจของการทำงานบนวินโดว์สนั่นเอง

ระบบการส่งเมสเสจบนวินโดว์เป็นแบบ Asynchronous นั่นหมายความว่า ไม่มีลำดับของเมสเสจที่แน่นอน เมสเสจแต่ละเมสเสจอาจจะถูกส่งมายังแอพพลิเคชันเมื่อไรก็ได้ วินโดว์ทุกวินโดว์จะมีฟังก์ชันในการตอบสนองเมสเสจ เมื่อได้รับเมสเสจจึงประมวลผลแล้วส่งคอนโทรลกลับคืนให้ระบบ หลังจากนั้นระบบอาจจะส่งเมสเสจไปให้วินโดว์อื่น ๆ ในแอพพลิเคชันเดียวกันนั้น หรือต่างแอพพลิเคชันก็ได้

เราจะเห็นได้อย่างชัดเจนว่า เหมือนกับว่าแอพพลิเคชันนั่งรอเมสเสจจากระบบ (ซึ่งจะมาเมื่อไรก็ได้) เมื่อเมสเสจถูกส่งมาทำการประมวลผลแล้วคืนคอนโทรลกลับไป วิธีการเขียนโปรแกรมแบบนี้เป็นหลักการเขียนโปรแกรมแบบที่เรียกว่า "Event-driven Programming" หรือ "Message Programming" ระบบ GUI อื่น ๆ ก็จะมีหลักการเขียนโปรแกรมแบบนี้ อาจจะแตกต่างบ้างก็รายละเอียดเท่านั้น (โปรแกรมเมอร์แมคอินทอชจะมีคำพูดที่ว่า "Don't call us, we'll call you")

การส่งเมสเสจ

อย่างไรก็ตาม มีเรื่องที่ต้องทำความเข้าใจให้มากเกี่ยวกับการส่งเมสเสจดังต่อไปนี้

1. เมสเสจถูกส่งมาได้สองวิธี วิธีแรกคือการส่งเมสเสจโดยผ่านคิว (PostMessage) และวิธีที่สองเป็นการส่งเมสเสจโดยตรงไม่ผ่านคิว (SendMessage) ความแตกต่างก็คือว่า ในวิธีการแรกเมสเสจจะถูกนำไปไว้ในคิวโดยการต่อท้ายเพื่อรอการประมวลผล ส่วนวิธีที่สองเป็นการเรียกไปที่ฟังก์ชันตอบสนองเมสเสจของวินโดว์นั้น ๆ โดยตรง ดังนั้นวิธีการที่สองจะเร็วกว่าวิธีการแรก

ในกรณีที่ PostMessage เพื่อส่งเมสเสจที่หนึ่งไปแล้วจึง SendMessage เพื่อส่งเมสเสจที่สองทีหลัง อาจเป็นไปได้ว่าเมสเสจที่สองอาจจะถูกประมวลผลก่อนเมสเสจที่หนึ่ง

2. เมสเสจสามารถซ้อนเมสเสจได้ ยกตัวอย่างเช่น เมื่อระบบส่งเมสเสจที่หนึ่งมาให้ฟังก์ชันที่ตอบสนองเมสเสจทำการ SendMessage เพื่อส่งเมสเสจที่สอง (ให้ตัวเอง) เมื่อจบการประมวลผลเมสเสจที่สองแล้วจึงกลับที่การประมวลผลเมสเสจที่หนึ่งต่อไป

ในกรณีที่ SendMessage เดียวกันไปให้ตัวเอง แอพพลิเคชันจะเกิดอาการสแต็กเต็ม แต่ถ้าเป็น PostMessage จะไม่เกิดสแต็กเต็ม แต่จะทำให้โปรแกรมรันไม่มีที่สิ้นสุด

3. ข้อนี้ต่อเนื่องจากหัวข้อที่แล้ว คือว่า เมื่อมีการส่งเมสเสจซ้อนเมสเสจได้ ดังนั้นฟังก์ชันที่ตอบสนองเมสเสจจะต้องมีลักษณะ "re-entered" ได้ และการใช้ตัวแปรแบบโกลบอลหรือสเตติกจะต้องระวังให้มาก

4. เมสเสจควรจะถูกประมวลผลโดยไม่ขึ้นต่อกัน นั่นคือไม่ควรจะมีกรณีที่การประมวลเมสเสจนี้ต้องผ่านการประมวลผลเมสเสจนั้นเสียก่อน เพราะว่าลำดับของเมสเสจไม่แน่นอนนั่นเอง (ที่สำคัญ ไมโครซอฟต์วินโดวส์เองอาจจะมีการเปลี่ยนโครงสร้างภายในในเวอร์ชันถัดไปก็ได้) ยกตัวอย่างเช่น เราคิดว่าก่อนเมสเสจ WM_GETMINMAXINFO ควรจะผ่านการประมวลผลเมสเสจ WM_CREATE เสียก่อน เมสเสจ WM_CREATE น่าจะถูกส่งมาก่อนตั้งแต่เริ่มสร้างวินโดว์ การทึกทักเช่นนี้ผิดมหันต์ เพราะเมื่อมีการสร้างวินโดว์ขึ้นมา เมสเสจ WM_CREATE ไม่ใช่เมสเสจแรกที่ส่งมาและจริง ๆ แล้วเมสเสจ WM_GETMINMAXINFO เป็นเมสเสจแรกที่ส่งมา

ฟังก์ชัน GetMessage และ PeekMessage

ที่กล่าวมานั้นเป็นหลักการทั่ว ๆ ไปของระบบการส่งเมสเสจของไมโครซอฟต์วินโดวส์ ผู้พัฒนาโปรแกรมบนวินโดวส์จะต้องตระหนักถึงหลักการดังกล่าวให้ดี การเขียนโปรแกรมโดยผิดหลักการจะทำให้เกิดการประมวลผลที่ผิดพลาดได้

การส่งเมสเสจสามารถกระทำได้โดยผ่านสองฟังก์ชันคือ PostMessage และ SendMessage ตามที่ได้กล่าวไปแล้ว ฟังก์ชัน SendMessage เป็นการส่งเมสเสจโดยตรง ดังนั้นวินโดว์ที่ได้รับเมสเสจจะประมวลผลทันที ส่วน PostMessage เป็นการนำเมสเสจไปไว้ในคิวเพื่อประมวลผลทีหลัง การประมวลผลนี้ทำได้อย่างไรจะได้กล่าวต่อไป

คิว

คิวคือที่ที่ใช้เก็บเมสเสจและอีเวนต์ (ต่างกันอย่างไรจะได้กล่าวต่อไป) ระบบวินโดวส์ประกอบด้วยคิวสองชนิดคือ คิวระบบและคิวแอพพลิเคชัน

คิวระบบ (System Queue)

คิวระบบใช้เก็บอีเวนต์ (Event) อีเวนต์คืออินพุตจากฮาร์ดแวร์นั่นเอง อันได้แก่ เม้าส์และคีย์บอร์ด เมื่อมีการกดเม้าส์หรือคีย์บอร์ด ไดรเวอร์ของเม้าส์หรือคีย์บอร์ดจะนำอีเวนต์ที่เกิดขึ้นไปเก็บไว้ในคิวระบบซึ่งจะอยู่ภายใน USER.EXE สำหรับเครื่องคอมพิวเตอร์

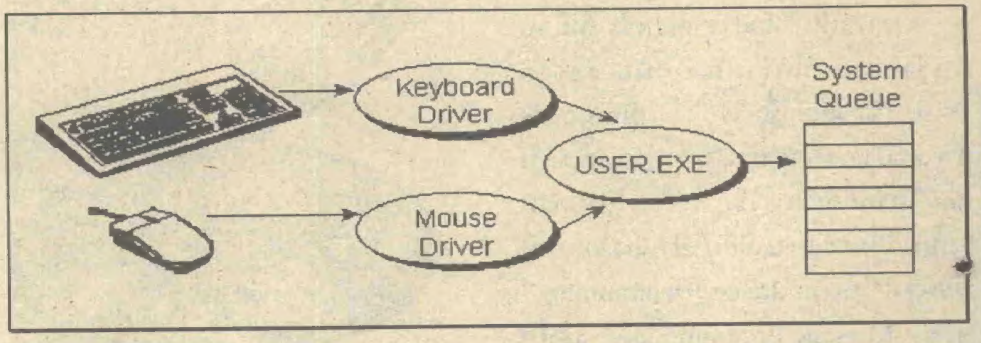
แบบ pen-computing จะมีการทำงานในลักษณะเดียวกัน ดังรูปที่ 1

สำหรับวินโดวส์ 3.1 คิวระบบสามารถเก็บอีเวนต์ได้จำกัด คือสูงสุดได้ 120 อีเวนต์ ในกรณีที่แอพพลิเคชันประมวลผลนาน ๆ และมีอีเวนต์เข้ามามากจนเต็มคิว ระบบจะส่งเสียงเตือนออกมา (beep)

คิวแอพพลิเคชัน (Application Queue)

คิวแอพพลิเคชันใช้เก็บเมสเสจ (message) เมสเสจได้มาจากอีเวนต์และเมสเสจของระบบเอง ทุก ๆ แอพพลิเคชันจะมีคิวของมันเอง คิวแอพพลิเคชันจะถูกสร้างเมื่อมีการสร้างแอพพลิเคชันในตอนเริ่มต้น ปกติแล้วแอพพลิเคชันจะวนลูปเพื่อนำเมสเสจออกจากคิวและส่งต่อไปยังวินโดว์ต่าง ๆ เมสเสจที่อยู่ในคิวแอพพลิเคชันจะต้องเกิดจากฟังก์ชัน PostMessage หรือ PostAppMessage เท่านั้น ส่วนฟังก์ชัน PostQuitMessage ไม่ได้ทำให้เกิดเมสเสจในคิวแอพพลิเคชัน (เมสเสจ WM_QUIT จะได้พูดถึงต่อไป)

เช่นเดียวกับคิวระบบคิวแอพพลิเคชันมีขนาดจำกัดคือ เก็บเมสเสจได้สูงสุด 8 เมสเสจเท่านั้น แต่เท่านี้ก็เป็นจำนวนเพียงพอ เพราะการส่งเมสเสจโดยส่วนใหญ่จะใช้ฟังก์ชัน SendMessage เมสเสจส่วนน้อยจะอยู่ภายในคิว อย่างไรก็ตามแอพพลิเคชันสามารถเปลี่ยนขนาดคิวของตนเองได้โดยใช้ฟังก์ชัน SetMessageQueue แต่ต้องทำด้วยความระมัดระวัง และต้องทำในช่วงที่ไม่มีเมสเสจในคิว คือ ตอนเริ่มแอพพลิเคชัน (ใน WinMain) หรือ ตอนที่ใช้ฟังก์ชัน PeekMessage แล้วพบว่าไม่มีเมสเสจในคิว การเปลี่ยนขนาดคิวโดยที่มีเมสเสจอยู่ในคิวจะทำให้เมสเสจหายไป



รูปที่ 1 การเกิดอีเวนต์และคิวระบบ

GetMessage และ PeekMessage ทำงานอย่างไร

จริง ๆ แล้วฟังก์ชัน GetMessage และ PeekMessage จะใช้โค้ดร่วมกัน ความแตกต่างที่สำคัญขึ้นอยู่กับช่วงที่ไม่มีเมสเสจภายในคิว โดยฟังก์ชัน GetMessage จะเซตให้แอพพลิเคชัน sleep ส่วน PeekMessage จะรีเทอร์นกลับไปยังแอพพลิเคชันด้วยค่าศูนย์ นอกจากนี้ก็มีความแตกต่างเล็ก ๆ น้อย ๆ อย่างอื่นดังจะได้กล่าวต่อไป

ขั้นตอนการทำงาน

รูปที่ 2 แสดงขั้นตอนการทำงานซึ่งมีรายละเอียดดังนี้

1. ตรวจสอบว่ามีเมสเสจในคิวแอพพลิเคชันหรือไม่ ถ้าผ่านให้ตรวจฟิลเตอร์ ถ้าผ่านฟิลเตอร์ให้กำจัดเมสเสจออกจากคิวแล้วรีเทอร์นกลับไปยังแอพพลิเคชัน ถ้าไม่ผ่านฟิลเตอร์ให้ทิ้งเมสเสจไว้ตามเดิม แล้วสแกนหาเมสเสจในคิวแอพพลิเคชันที่ผ่านฟิลเตอร์ แล้วรีเทอร์นกลับไป

2. ถ้าไม่มีเมสเสจในคิวแอพพลิเคชันให้ทำการสแกนคิวระบบที่อีเวนต์แรกต้นคิวที่เป็นของแอพพลิเคชันนั้น แล้วทำการเปลี่ยนอีเวนต์ให้เป็นเมสเสจส่งกลับไปยังแอพพลิเคชันโดยไม่ผ่านคิว

3. ถ้าไม่มีอีเวนต์ในคิวระบบ ให้ทำการตรวจสอบ update region ของทุกวินโดว์ในแอพพลิเคชัน ถ้าพบว่ามี (นั่นคือต้องการ paint) ให้สร้าง WM_PAINT แล้วรีเทอร์นไปยังแอพพลิเคชัน (สังเกตว่า WM_PAINT จะไม่อยู่ในคิว)

4. ถ้าไม่มี update region ให้ทำการตรวจสอบว่าต้องคืนคอนโทรลหรือไม่ (ไม่คืนกรณีเดียวคือเรียกฟังก์ชัน PeekMessage ด้วย PM_NOYIELD) ถ้าใช่ให้ทำการคืนคอนโทรล (yielding) ถ้าไม่ให้ทำขั้นต่อไป

5. เมื่อมาถึงขั้นนี้ให้ตรวจว่ามีวินโดว์ใดในแอพพลิเคชันที่ต้องการไทเมอร์ และถึงเวลาที่ต้องส่งเมสเสจไทเมอร์แล้วหรือยัง ถ้าใช่ให้สร้างเมสเสจ WM_TIMER แล้วส่งกลับยังแอพพลิเคชัน ถ้าไม่ให้ทำขั้นต่อไป

6. ทำการตรวจสอบว่าจบแอพพลิเคชันแล้วหรือยัง ถ้าจบแล้วให้เคลียร์ข้อมูลของแอพพลิเคชันใน GDI (โดยใช้ฟังก์ชัน LocalShink) แล้วส่ง WM_QUIT ไปให้แอพพลิเคชัน ถ้าไม่ให้ทำขั้นต่อไป

7. ถึงจุดนี้เป็นจุดที่ต้องแยกว่าเป็นฟังก์ชัน GetMessage หรือ PeekMessage เรียกเข้ามา

7.1 ถ้าเป็น GetMessage ให้เซตแอพพลิเคชัน sleep รอให้เกิดเมสเสจขึ้นมาจึงจะปลุกแอพพลิเคชันขึ้นมา แล้วกลับไปยังขั้นตอนที่หนึ่ง

7.2 ถ้าเป็น PeekMessage มี 2 กรณีคือ PeekMessage ถูกเรียกเข้ามาด้วย PM_NOYIELD หรือไม่ ถ้าใช่ให้รีเทอร์นศูนย์กลับไปยังแอพพลิเคชัน ถ้าไม่ให้คืนคอนโทรลก่อนจะรีเทอร์นกลับไปยังแอพพลิเคชัน

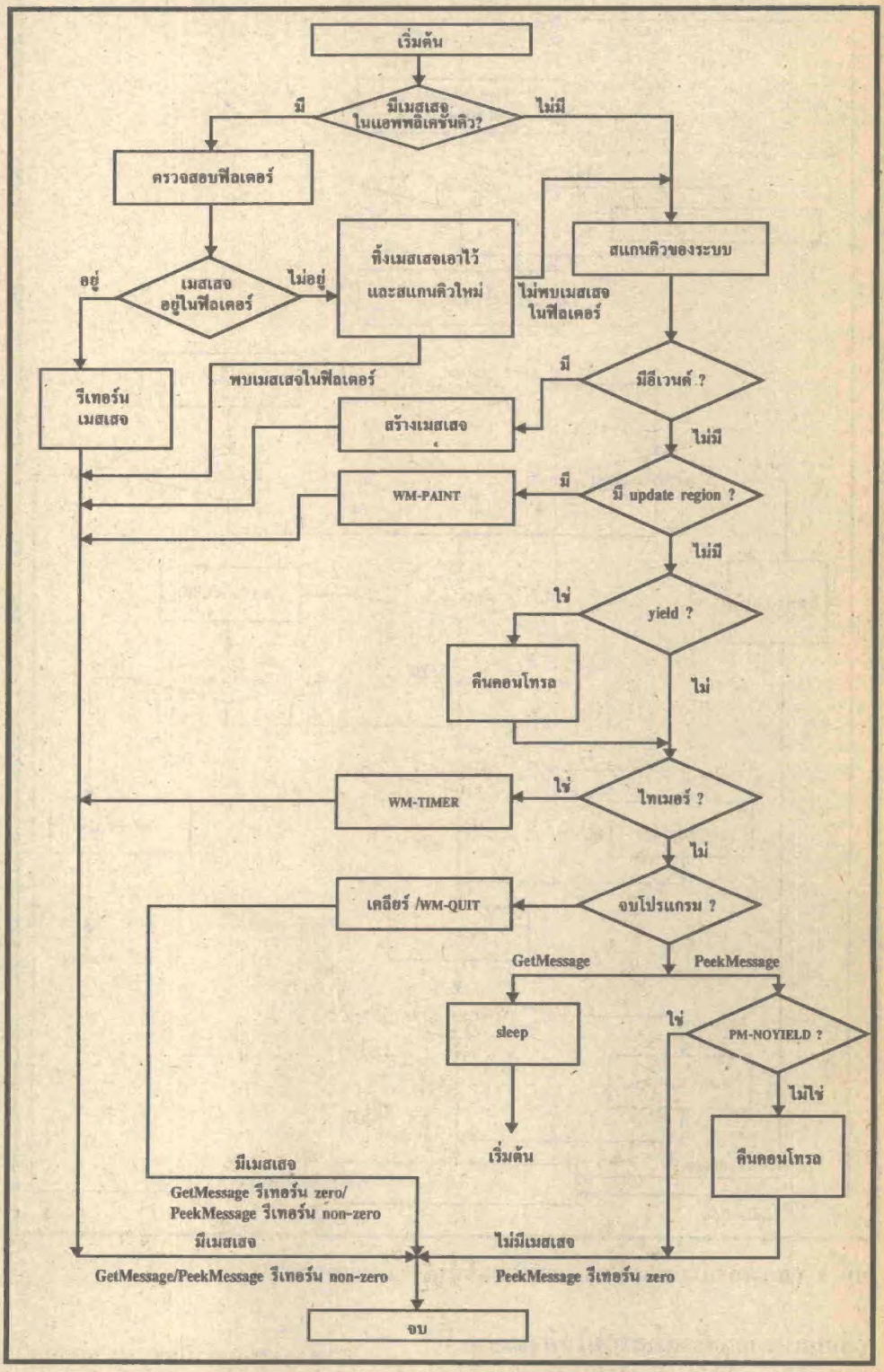
WH_GETMESSAGE

ทุกครั้งที่ GetMessage และ PeekMessage จะรีเทอร์นกลับไปยังแอพพลิเคชัน ระบบจะตรวจสอบว่ามีการดักเมสเสจหรือไม่ ถ้ามีจะเรียกไปที่ฟังก์ชันที่ดักเสียก่อน (ยกเว้นกรณีที่ PeekMessage รีเทอร์นค่าศูนย์)

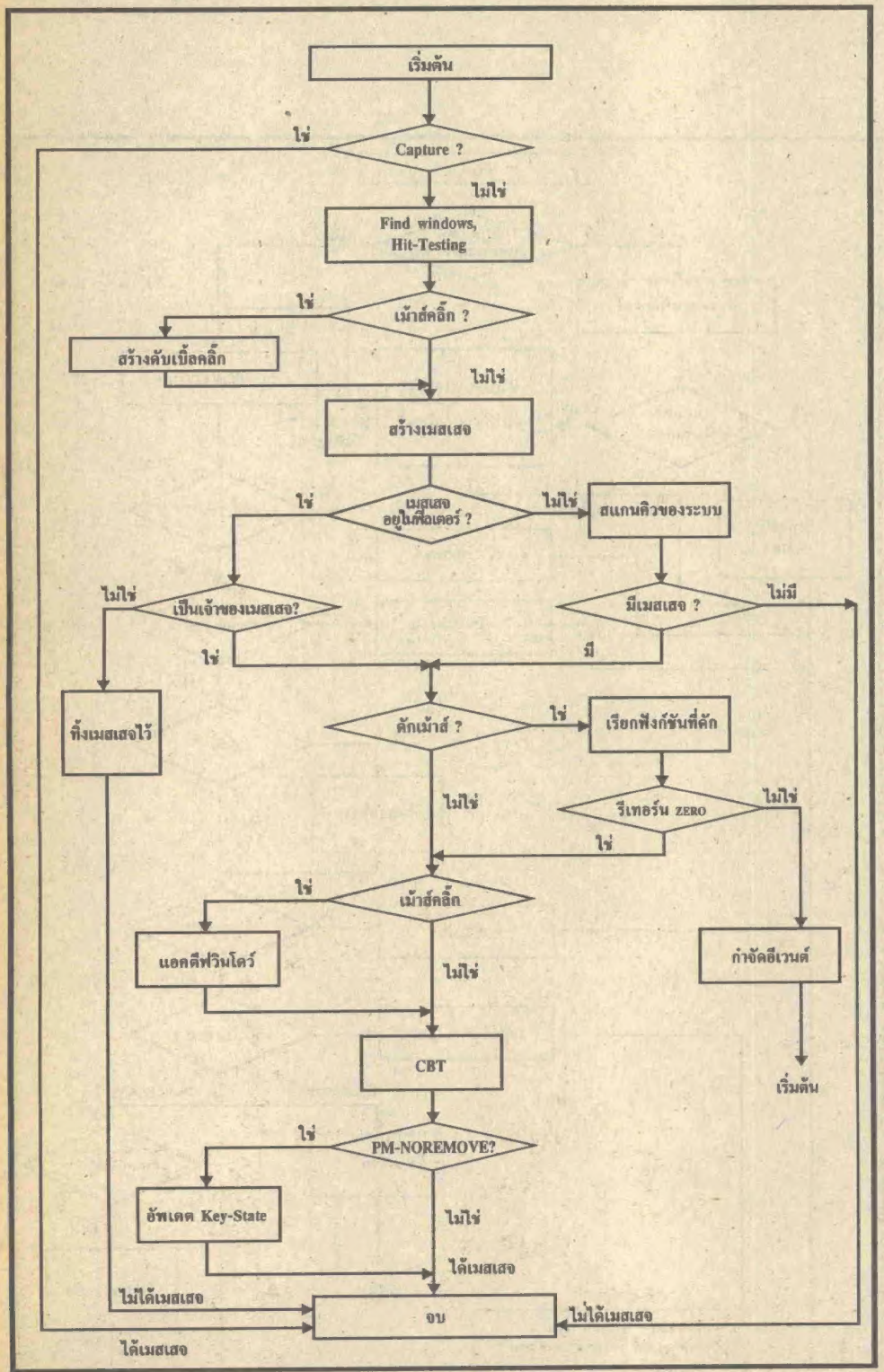
การสแกนระบบคิว

จากหัวข้อที่แล้วท่านผู้อ่านคงจะเข้าใจการทำงานของฟังก์ชัน GetMessage และ PeekMessage พอสมควรแล้ว ภายในการทำงานดังกล่าวได้มีการกล่าวถึงการสแกนคิวระบบ ซึ่งเป็นการค้นหาอีเวนต์จากฮาร์ดแวร์เพื่อนำมาประมวลผลอีกที เราจะกล่าวถึงรายละเอียดต่อไป

การสแกนคิวระบบเกิดขึ้นเมื่อไม่มีเมสเสจในคิวแอพพลิเคชัน การสแกนเริ่มด้วยการตรวจสอบอีเวนต์แรกของคิวระบบ เนื่องจากว่าอีเวนต์มี 2 ชนิดคือ อีเวนต์จากเม้าส์และอีเวนต์จากคีย์บอร์ด การสแกนหาอีเวนต์ทั้งสองจึงแยกทำงานกันโดยเด็ดขาด



รูปที่ 2 แสดงการทำงานภายในฟังก์ชัน GetMessage และ PeekMessage



รูปที่ 3 แสดงการทำงานของการสแกนอีเวนต์เม้าส์ภายในคิวระบบ

การสแกนหาอีเวนต์เม้าส์ภายในคิวระบบ

ขั้นตอนมีดังนี้ (ดูรูปที่ 3 ประกอบ)

1. ตรวจสอบว่ามีการ "capture" เม้าส์หรือไม่ ถ้ามีให้นำอีเวนต์ออกจากคิว แปลงให้เป็นเมสเสจแล้วรีเทอร์นกลับไปโดยมีเมสเสจ

2. ถ้าไม่มีการ "capture" ให้ทำการตรวจสอบว่าอีเวนต์นั้นเป็นของวินโดว์ไหนและแอพพลิเคชันอะไร พร้อมทำ Hit-Testing ซึ่งเป็นการตรวจสอบว่า เม้าส์พอยน์เตอร์อยู่บนไทเทิ้ลบาร์ คอนโทรลเมนูบ็อกซ์...หรือเป็นไคล์เอ็นต์แอเรีย เป็นต้น

3. หลังจากนั้นให้ตรวจสอบว่าอีเวนต์นั้นเป็นการคลิ๊ก (กดเม้าส์ลง-เม้าส์ดาวน์) หรือไม่ ถ้าใช่ให้ตรวจสอบว่าต้องสร้างดับเบิ้ลคลิ๊กหรือไม่

4. สร้างเมสเสจจากอีเวนต์นั้น

5. ขั้นตอนนี้เป็นการตรวจว่าเมสเสจอยู่ในฟิลเตอร์หรือไม่ ถ้าใช่แต่เมสเสจไม่เป็นของแอพพลิเคชันให้ทิ้งอีเวนต์ไว้ในคิวระบบตามเดิม และรีเทอร์นกลับโดยไม่มีเมสเสจ ถ้าเมสเสจอยู่ในฟิลเตอร์และเป็นของแอพพลิเคชันนั้นให้ไปขั้นตอนที่ 7

6. ในกรณีที่เมสเสจไม่อยู่ในฟิลเตอร์ให้ทำการสแกนคิวระบบเพื่อหาอีเวนต์ถัดไป ถ้ามีอีเวนต์ให้ไปขั้นตอนที่ 7 ถ้าไม่มีเลยให้รีเทอร์นโดยไม่มีเมสเสจ

7. ตรวจสอบว่ามีการดักเม้าส์หรือไม่ ถ้ามีให้เรียกไปที่ฟังก์ชันที่ดัก ถ้าฟังก์ชันรีเทอร์นศูนย์ให้ไปขั้นตอนที่ 8 ถ้ารีเทอร์น non-zero ให้กำจัดอีเวนต์ออกจากคิว แล้วกลับไปเริ่มต้นใหม่

8. ขั้นตอนนี้เป็นการตรวจสอบว่าเป็นการคลิ๊กเม้าส์หรือไม่ ถ้าใช่ให้ทำการแอคติฟวินโดว์ดังนี้

8.1 หาวินโดว์หลัก (top-level) ของวินโดว์นั้น

8.2 ส่งเมสเสจ WM_MOUSE-ACTIVATE โดยใช้ฟังก์ชัน Send-Message

8.3 ตรวจสอบค่าที่รีเทอร์นกลับมาดังนี้

- ถ้าค่ารีเทอร์นเป็น NULL, MA_ACTIVATE หรือ MA_ACTIVATEANDEAT ให้เรียกฟังก์ชัน ActiveWindow เพื่อทำการแอคทิเวทวินโดว์หลักนั้น
- ถ้าค่ารีเทอร์นเป็น MA_NOACTIVATE หรือ MA_NOACTIVATEANDEAT แอพพลิเคชันจะไม่ถูกแอคทิเวท
- เมสเสจ WM_SETCURSOR ถูกส่งไปให้วินโดว์หลักนั้นเพื่อเซตรูปร่างของเม้าส์พอยน์เตอร์

9. ขั้นตอนนี้เป็นการตรวจสอบว่ามีการติดตั้ง CBT หรือไม่ ถ้ามีการติดตั้งให้เรียกไปด้วยรหัส HCBT_CLICKSKIPPED

10. ต่อไปเป็นการตรวจสอบว่าเป็นการเรียก PeekMessage ด้วย PM_NOREMORE หรือไม่ ถ้าใช่ให้ทำการอัพเดตตาราง key-state และกำจัดอีเวนต์ออกจากคิวระบบ ถ้าไม่ใช่ไม่ต้องอัพเดตตาราง key-state และไม่ต้องกำจัดอีเวนต์

11. รีเทอร์นกลับด้วยเมสเสจที่ได้มา

การสแกนอีเวนต์คีย์บอร์ดในคิวระบบ

ขั้นตอนมีดังนี้ (ดูรูปที่ 4 ประกอบ)

1. ตรวจสอบว่าอีเวนต์นั้นเป็น CTRL+ESC หรือไม่ ถ้าใช่ให้เรียก Task List แล้วกลับไปเริ่มต้นใหม่ การเรียก Task List ทำได้โดยส่งเมสเสจ WM_SYSCOMMAND (wParam เป็น SC_TASKLIST) ไปให้วินโดว์ที่แอคตีฟอยู่ การเรียก Task List จะไม่เกิดขึ้นถ้าวินโดว์ที่แอคตีฟอยู่เป็น system modal เช่น MessageBox ที่มีเงื่อนไขแบบ MB_ICONHAND และ MB_SYSTEMMODAL

2. ตรวจสอบว่าอีเวนต์เป็นการทำ PRINT SCREEN หรือไม่ ถ้าใช่ให้จับภาพจากจอ (ทั้งจอหรือเฉพาะวินโดว์ที่แอคตีฟอยู่) ลงคลิปบอร์ด ถ้ามีการกดคีย์ ALT จะจับภาพเฉพาะวินโดว์ที่แอคตีฟ ถ้าไม่จะจับทั้งจอ เช่นเดียวกับขั้นตอนที่ 1 ถ้าวินโดว์ที่แอคตีฟเป็น system modal จะไม่มีการจับภาพลงคลิปบอร์ด

3. ตรวจสอบ Hot-Key ถ้าใช่ให้ส่งเมสเสจ (แบบ post) WM_SYSCOMMAND (wParam เป็น SC_HOTKEY) ไปให้วินโดว์ที่แอคตีฟอยู่ ถ้าวินโดว์ที่แอคตีฟอยู่เป็น system modal ให้ยกเลิกการทำงานดังกล่าว

4. ถึงขั้นตอนนี้เราจะได้อีเวนต์จากคีย์บอร์ดแล้ว

5. ตรวจสอบว่าอีเวนต์เป็นของแอพพลิเคชันหรือไม่ ถ้าไม่ให้ทิ้งอีเวนต์ไว้ตามเดิม แล้วรีเทอร์นกลับไปแบบไม่มีเมสเสจ โดยให้ทำการปลุกแอพพลิเคชันที่เป็นเจ้าของขึ้นมาด้วย

6. ต่อไปให้ทำการตรวจสอบโฟกัส ถ้าไม่มีวินโดว์ใดได้โฟกัสให้สร้างเมสเสจซีสเต็มส์คีย์ (เช่น WM_SYSKEYUP, WM_SYSCHAR เป็นต้น) ส่งไปให้วินโดว์ที่แอคตีฟ

7. ตรวจสอบว่าเมสเสจอยู่ในฟิลเตอร์หรือไม่ ถ้าไม่ให้สแกนคิวระบบเพื่อหาอีเวนต์ต่อไป ถ้าเจอให้ทำขั้นตอนที่ 8 ถ้าไม่มีให้รีเทอร์นกลับและไม่มีเมสเสจ

8. นำอีเวนต์ออกจากคิวระบบ ยกเว้นในกรณีที่เรียกมาด้วยฟังก์ชัน PeekMessage ด้วย PM_NOREMOVE

9. ตรวจสอบว่ามีการดักเมสเสจคีย์บอร์ดหรือไม่ ถ้าใช่ให้เรียกฟังก์ชันดังกล่าว

10. ตรวจสอบว่ามีการติดตั้ง CBT หรือไม่ ถ้ามีให้เรียกฟังก์ชันดังกล่าวด้วย HCBT_KEYSKIPPED

11. รีเทอร์นกลับด้วยเมสเสจ

PeekMessage และ PM_NOREMOVE

ปกติแล้ว PeekMessage และ GetMessage จะกำจัดอีเวนต์และเมสเสจออกจากคิว ยกเว้นกรณีที่เรียก PeekMessage ด้วย PM_NOREMOVE จะได้เมสเสจกลับมาแต่ไม่นำเมสเสจออกจากคิว ประโยชน์ของการทำเช่นนี้พอจะมีอยู่บ้างเช่น ต้องการตรวจสอบว่ามีเมสเสจนั้น ๆ อยู่ในคิวหรือไม่ หรือต้องการทำงานแบบแบคกราวนด์

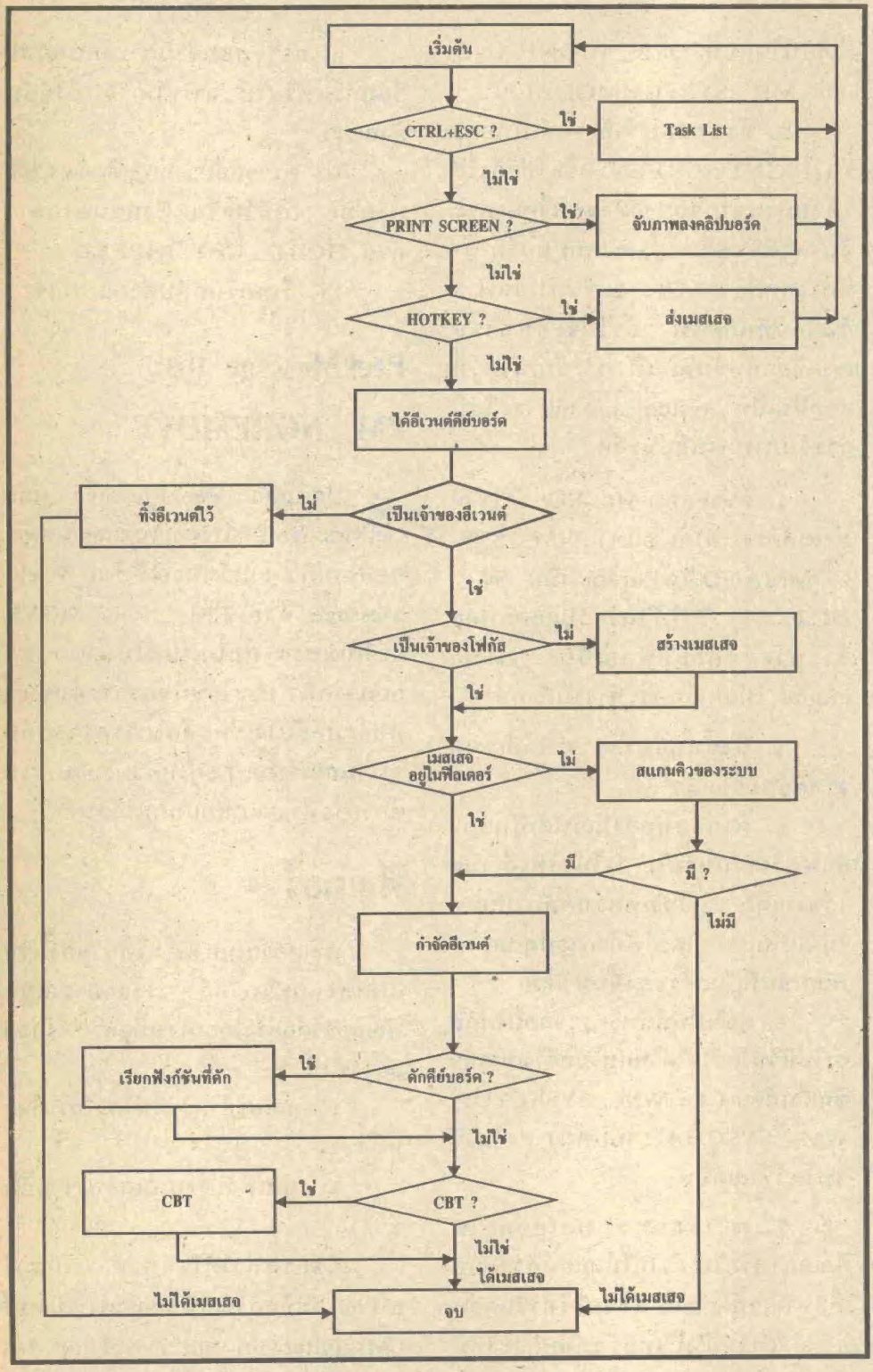
ฟิลเตอร์

ฟิลเตอร์เป็นการตั้งเงื่อนไขสำหรับเมสเสจและวินโดว์ เราสามารถเซตฟิลเตอร์ได้อย่างใดอย่างหนึ่งหรือทั้งสองอย่างดังนี้

1. เมสเสจสำหรับวินโดว์ใดวินโดว์หนึ่ง

2. เมสเสจบางเมสเสจ (เซตเป็นช่วง)

วิธีแรกทำได้โดยส่งพารามิเตอร์ hWnd วิธีที่สองทำได้โดยส่งพารามิเตอร์ uMsgFilterMin และ uMsgFilterMax พารามิเตอร์ดังกล่าวเป็นพารามิเตอร์ของฟังก์ชัน GetMessage และ PeekMessage ซึ่งปกติแล้ว เรามักจะไม่เซตฟิลเตอร์ ซึ่งทำได้โดยส่งพารามิเตอร์ hWnd เป็น NULL และส่งพารามิเตอร์



รูปที่ 4 แสดงการทำงานของการสแกนอีเวนต์คีย์บอร์ดภายในคิวระบบ

uMsgFilterMin และ uMsgFilterMax เป็น NULL โปรดจำไว้ว่าเฉพาะเมสเสจ เม้าส์และคีย์บอร์ด เมสเสจที่ส่งโดยฟังก์ชัน PostMessage, เมสเสจ WM_PAINT, และเมสเสจไทเมอร์ (WM_TIMER) เท่านั้นที่ทำฟิลเตอร์ได้ เพราะเมสเสจส่วนใหญ่จะถูกส่งด้วยฟังก์ชัน SendMessage

ปัญหาที่เกิดกับฟิลเตอร์

เราจะต้องใช้ฟิลเตอร์ด้วยความระมัดระวังเพราะอาจก่อให้เกิดปัญหาดังต่อไปนี้ได้ ปกติเมสเสจจะเข้ามาแบบอนุกรม แอพพลิเคชันพอจะคาดเดาได้ว่าเมื่อมีการกดเม้าส์ ก็ต้องมีการปล่อยเม้าส์เป็นต้น ถ้ามีการเซตฟิลเตอร์และมีการกำจัดเมสเสจอาจเป็นไปได้ว่าแอพพลิเคชันจะได้เม้าส์ดาวน์แต่ไม่ได้เม้าส์อัพ ดังนั้นจึงมีข้อแนะนำสำหรับการเซตฟิลเตอร์ดังนี้คือ

1. อย่าเซตฟิลเตอร์สำหรับเมสเสจจากเม้าส์และคีย์บอร์ด
2. ถ้าเซตฟิลเตอร์สำหรับคีย์บอร์ดหรือเม้าส์ อย่ากำจัดเมสเสจ (ใช้ PeekMessage และ PM_NOREOMOVE)

WM_QUIT

เมื่อตอนต้นบทความได้กล่าวไปแล้วว่า PostQuitMessage ไม่ได้ทำให้เกิด WM_QUIT ในคิว และจากรายละเอียดของการทำงานของฟังก์ชัน GetMessage และ PeekMessage ที่แสดงให้เห็นชัดเจนแล้วว่าเมสเสจ WM_QUIT จะไม่อยู่ในคิว

จากไดอะแกรมรูปที่ 2 จะเห็นว่าเมื่อเกิดเมสเสจ WM_QUIT ฟังก์ชัน GetMessage จะรีเทอร์นศูนย์ แต่ฟังก์ชัน PeekMessage จะรีเทอร์น non-zero แอพพลิเคชันที่ใช้ PeekMessage จะต้องตรวจเมสเสจ WM_QUIT นี้ด้วย ฟังก์ชัน PostQuitMessage จริง ๆ แล้วไม่ได้ทำอะไรนอกจากเซตแฟล็กเอาไว้ เมื่อไม่มีเมสเสจเหลืออยู่ในคิว

แล้วฟังก์ชัน GetMessage และ PeekMessage จะตรวจสอบแฟล็กนี้และรีเทอร์นเมสเสจ WM_QUIT กลับไป

ความแตกต่างระหว่างการคืนคอนโทรล (yielding) และ sleep

ระบบวินโดว์เป็นระบบมัลติทาสก์กิ้งแบบนอนพรีเอมทีฟ (Non-preemtive MultiTasking) ซึ่งหมายความว่าสามารถรันแอพพลิเคชันได้พร้อม ๆ กัน ดังนั้นระบบจึงต้องมีการจัด schedule ระบบวินโดวส์จัด schedule โดยใช้ TDB List ซึ่งเป็นลิสต์ของ TDB แต่ละ TDB จะเก็บข้อมูลต่าง ๆ เกี่ยวกับแอพพลิเคชันนั้น ๆ (หนึ่ง TDB ต่อหนึ่งแอพพลิเคชัน) ภายใน TDB มีข้อมูลสำคัญอันหนึ่งคือ สภาวะของแอพพลิเคชันซึ่งมีอยู่สองสภาวะคือ ready to run และ asleep

ระบบวินโดว์จะดึงแอพพลิเคชันจาก TDB List มาทีละแอพพลิเคชันเพื่อทำการรันแอพพลิเคชันนั้น โดยหมุนเวียนให้ครบ TDB List จนครบรอบแล้วเริ่มต้นใหม่ การหมุนเวียนเลือกแอพพลิเคชันมารันนั้น จะเลือกเฉพาะ TDB ที่มีสภาวะเป็นแบบ ready to run เท่านั้น

ฟังก์ชัน Yield ทำหน้าที่หยุดการรันแอพพลิเคชัน และส่งการรันไปยังแอพพลิเคชันถัดไปใน TDB List แต่จะไม่เปลี่ยนสภาวะจาก ready to run เป็น asleep แอพพลิเคชันที่เรียกฟังก์ชัน Yield ยังคงมีสภาวะเป็น ready to run

ปัญหาของฟังก์ชัน Yield

ปัญหาของการใช้ฟังก์ชัน Yield เกิดจากข้อจำกัดที่ว่าอีเวนต์ในระบบเป็นแบบอนุกรม ถ้าแอพพลิเคชันกำลังทำงานที่กินเวลานาน ๆ ในระหว่างนี้จะไม่มีแอพพลิเคชันอื่นได้รัน และอาจมีการกดคีย์บอร์ดหรือเม้าส์ได้ ซึ่งจะทำให้เกิดเหตุการณ์ดังนี้

1. ในระหว่างที่ทำงานนี้แอพพลิเคชันเรียกใช้ฟังก์ชัน Yield
2. การรันย้ายจากแอพพลิเคชันนี้ไปยังแอพพลิเคชันอื่นที่มีสภาวะ ready to run
3. แอพพลิเคชันใหม่ (ซึ่งควรจะอยู่ในฟังก์ชัน GetMessage) ตรวจสอบคิวของมันพบว่าคิวของมันไม่มีเมสเสจจึงทำการตรวจสอบคิวระบบ
4. จากการตรวจสอบคิวระบบพบว่ามีอีเวนต์ที่ต้นคิว แต่เป็นของแอพพลิเคชันก่อนหน้านี้ ดังนั้นแอพพลิเคชันจึง sleep แล้วกลับไปที่แอพพลิเคชันก่อนหน้านี้ที่เรียก Yield
5. แอพพลิเคชันที่เรียกฟังก์ชัน Yield ทำการรันโค้ดถัดจาก Yield โดยที่มีการโยกย้ายไปแต่ไม่มีการรันเลย

โดยสรุปแล้วปัญหาของ Yield เกิดขึ้นเมื่อมันไม่ยอมให้แอพพลิเคชันอื่นประมวลผลคิวระบบ ดังนั้นไม่ควรใช้ Yield ถ้าในคิวระบบมีอีเวนต์ของแอพพลิเคชันนั้น

WaitMessage

ฟังก์ชัน WaitMessage ทำหน้าที่ง่าย ๆ คือ ทำการ sleep แอพพลิเคชันสภาวะจะเปลี่ยนไปเป็น asleep แอพพลิเคชันใช้ฟังก์ชัน WaitMessage เพื่อทำงานแบบแบคกราวนด์ดังรูปที่ 5

```
if (PeekMessage(...) != NULL)
        // Translate & dispatch the message
else if (there is background processing to perform)
        // Perform background processing
else
        // No background processing, no message - go to sleep
        WaitMessage( );
```

รูปที่ 5 ฟังก์ชัน WaitMessage ที่ทำการเปลี่ยน Sleep เป็น asleep เพื่อทำงานแบบแบคกราวนด์

จะเห็นได้ว่าแอพพลิเคชันจะเรียกฟังก์ชัน WaitMessage ก็ต่อเมื่อไม่มีเมสเสจและไม่มีการทำงานแบบแบคกราวนด์ เหตุผลที่ทำเช่นนี้คือ จะช่วยให้ระบบมีเวลาไปทำงานอย่างอื่น เช่น ทำ flush ดิสเกตต์หรือตรวจสอบแบตเตอรีของระบบ

สรุป

สำหรับเทคนิคการเขียนโปรแกรมบนวินโดว์ในตอนแรกนี้ก็ได้นำเสนอเรื่องเกี่ยวกับเมสเสจ ท่านผู้อ่านก็คงได้รู้และเข้าใจการทำงานภายในของฟังก์ชัน GetMessage และ PeekMessage บ้างพอสมควร การได้รู้ถึงการทำงานข้างในจะช่วยให้เราสามารถเขียนโปรแกรมได้ถูกต้องและมีประสิทธิภาพมากขึ้น หรือสามารถหาทางเลือกที่เหมาะสมกับงานของเราได้ ในตอนต่อ ๆ ไปจะได้นำเสนอเรื่องของเมสเสจที่สำคัญ ๆ จำเพาะเจาะจงไป สำหรับฉบับนี้ขอลาไปก่อนครับ